REMARKS

Claims 2, 3, 28 and 29 have been amended for clarity. No claims have been added or canceled. Therefore, claims 1-39 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Section 102(e) Rejection:

The Examiner rejected claims 1-7, 9, 10, 14-20, 22, 23, 27-33, 35 and 36 under 35 U.S.C. § 102(e) as being anticipated by Arthur Lee, et al. ("Building a Persistent Object Store using the Java Reflection API") (hereinafter "Lee"). Applicant traverses the rejection for at least the following reasons.

Claim 1

In regard to claim 1, the Examiner has not full and clearly stated her rejection since the Examiner failed to clearly indicate the specific portions of Lee that she considers to be equivalent to the *one or more classes* and the *one or more enhanced classes* of Applicant's claim. Applicant also notes the Examiner has failed to clearly state the specific portion of Lee that she considers to be equivalent to the *persistence structure specifying data fields of the one or more classes to be persisted* of Applicant's claim. Applicant notes MPEP 707.07(d), which requires that an Examiner's ground of rejection be "fully and clearly stated." Accordingly, Applicant respectfully requests that, for each particular element of Applicant's claim, the Examiner explicitly specify the element of Lee that she considers to be equivalent to the particular element of Applicant's claim. As the Examiner is certainly aware, anticipation requires the presence in a single prior art reference disclosure of each and every element of the claimed invention, arranged as in the claim. *Lindemann Maschinenfabrik GmbH v. American Hoist & Derrick Co.*, 221 USPQ 481, 485 (Fed. Cir. 1984) (emphasis added).

Applicant's note that claim 1 draws a distinction between the claimed *one or more classes* and the claimed *one or more enhanced classes*. While the portions of

Lee cited by the Examiner refer to two types of classes (i) a "`PersistentRoot` class" and (ii) "a class that inherits the `PersistentRoot` class," these classes do not meet the Applicant's specific claim limitations for *one or more classes* and *one or more enhanced classes*. For instance, the Examiner cites the following portions of Li:

> A persistent object created from a persistent class that inherits the `PersistentRoot` class is registered in the POS [persistent object store] system as persistent by the constructor of the class (Lee, page 2, column 2, paragraph 4).
>
> All the objects created from a class that inherits the `PersistentRoot` class will be persistent (Lee, page 3, column 1, paragraph 2).

Presumably, the Examiner considers the "`PersistentRoot` class" to be equivalent to Applicant's claimed *one or more classes* and a "class that inherits the `PersistentRoot` class" to be equivalent to Applicant's claimed *one or more enhanced classes*. (For instance, since Applicant's claim includes *an object of the one or more enhanced classes* and the only *objects* cited by the Examiner in the rejection of claim 1 are objects "created from a class that inherits the `PersistentRoot` class," the Examiner presumably equates a "class that inherits the `PersistentRoot` class" to the *one or more enhanced classes* of Applicant's claim. No other interpretation is apparent from the Examiner's rejection.) **As demonstrated below, Lee's "`PersistentRoot` class" and a "class that inherits the `PersistentRoot` class" fail to meet the specific limitations of Applicant's claimed *one or more classes* and *one or more enhanced classes*, respectively.**

Accordingly, Lee fails to teach a class structure based data object enhancer configured to (a) analyze the structure of the one or more classes to *determine a persistence structure* specifying data fields of the one or more classes to be persisted, **and (b)** generate one or more enhanced classes corresponding to the one or more classes such that an object of the one or more enhanced classes is enhanced to persist data of the data fields to be persisted *according to the persistence structure*, wherein said data of the data fields to be persisted is data of said object. Nowhere does Lee teach analyzing the structure of the `PersistentRoot` class (which the Examiner presumably equates to

Applicant's claimed *one or more classes)*, much less analyzing the structure of the `PersistentRoot` class to determine a persistence structure specifying data fields of the `PersistentRoot` class to be persisted. The Examiner fails to cite any portion of Lee that is equivalent to Applicant's *persistence structure*.

Furthermore, since Lee fails to teach determining a persistence structure specifying data fields of the `PersistentRoot` class (which, as described above, the Examiner presumably equates to Applicant's claimed *one or more classes)*, by extension **Lee cannot teach a class structure based data object enhancer configured to generate one or more enhanced classes corresponding to the one or more classes such that an object of the one or more enhanced classes is enhanced to persist data of the data fields to be persisted *according to the persistence structure*, wherein said data of the data fields to be persisted is data of said object.** Furthermore, in regard to this limitation, the Examiner cites the portion of Lee that reads, "All the objects created from a class that inherits the `PersistentRoot` class will be persistent" (Lee, page 3, column 1, paragraph 2). Presumably, the Examiner considers the *inheriting* described by Lee to teach the *generat[ion] of one or more enhanced classes* of Applicant's claim. **However, nowhere does Lee teach that inheritance includes generating one or more enhanced classes corresponding to the one or more classes such that an object of the one or more enhanced classes is enhanced to persist data of the data fields to be persisted *according to the persistence structure*.** One skilled in the art of computer science would recognize that inheritance is a basic tenet of object-oriented programming. No one of ordinary skill in the art would confuse inheritance with the specific type of generation recited in Applicant's claim.

Thus, for at least the reasons presented above, the rejection of claim 1 is unsupported by the cited art and removal thereof is respectfully requested. Similar remarks apply to claims 14 and 27.

Claim 2

In regard to claim 2, Lee fails to teach wherein to analyze the structure of the one or more classes, the class structure based enhancer is configured to make one or more Java reflection calls *to the one or more classes.* For at least the reasons presented above with respect to claim 1, it appears that the Examiner is relying on the `PersistentRoot` class to teach Applicant's claimed *one or more classes*. However, nowhere does Lee teach or suggest making one or more Java reflection calls *to the* `PersistentRoot` *class*. The Examiner cites "Java reflection API" of the section of Lee entitled "The life cycle of an object in the POS environment." The cited portion of Lee describes utilizing the Java reflection API to determine meta information about an object, not the `PersistentRoot` class (on which the Examiner presumably relies to teach the *one or more classes* of Applicant's claim).

Thus, for at least the reasons presented above, the rejection of claim 2 is unsupported by the cited art and removal thereof is respectfully requested. Similar remarks apply to claims 15 and 28.

Claim 4

In regard to claim 4, Lee fails to teach wherein the class structure based enhancer is further configured to generate metadata that includes results of the analysis of the structure *of the one or more classes.* For at least the reasons presented above with respect to claim 1, it appears that the Examiner is relying on the `PersistentRoot` class to teach Applicant's claimed *one or more classes*. However, nowhere does Lee teach or suggest generating metadata that includes the results of an analysis of *the* `PersistentRoot` *class*. The Examiner cites the following portion of Lee:

> In our design, however, all a persistent class has to do is inherit the `PersistentRoot` class. Then the meta information that we need to save an object can be obtained through the Java reflection API at run time. (page 2, column 2, paragraph 3)

Presumably the examiner considers the "meta information" taught by Lee to be the results of the analysis of the structure *of the one or more classes* of Applicant's claim. However, nowhere does Lee teach that such "meta information" is the result of an analysis of the `PersistentRoot` class (on which the Examiner relies to teach the *one or more classes* of Applicant's claim). In fact, as described above with respect to claim 2, Lee describes utilizing the Java reflection API to determine "meta information" about an object, not the `PersistentRoot` class (on which the Examiner presumably relies to teach the *one or more classes* of Applicant's claim). Accordingly, Lee fails to teach the specific limitations of Applicant's claim.

Thus, for at least the reasons presented above, the rejection of claim 4 is unsupported by the cited art and removal thereof is respectfully requested. Similar remarks apply to claims 17 and 30.

Claim 9

In regard to claim 9, Lee fails to teach wherein to determine a persistence structure for the data of the one or more classes the class structure based enhancer is configured to apply one or more rules to the results of Java reflection calls to or byte code parsing *of the one or more input classes.* For at least the reasons presented above with respect to claim 1, it appears that the Examiner is relying on the `PersistentRoot` class to teach Applicant's claimed *one or more classes*. However, nowhere does Lee teach or suggest applying one or more rules to the results of Java reflection calls to or byte code parsing *of the* `PersistentRoot` *class*. The Examiner cites the following portion of Lee:

> In our design, however, all a persistent class has to do is inherit the `PersistentRoot` class. Then the meta information that we need to save an object can be obtained through the Java reflection API at run time. (page 2, column 2, paragraph 3)

However, as described above, Lee describes utilizing the Java reflection API to determine meta information about an object, not the `PersistentRoot` class (on which the

Examiner presumably relies to teach the *one or more input classes* of Applicant's claim). Accordingly, Lee fails to teach the specific limitations of Applicant's claim.

Thus, for at least the reasons presented above, the rejection of claim 9 is unsupported by the cited art and removal thereof is respectfully requested. Similar remarks apply to claims 22 and 35.

Section 103(a) Rejection:

The Examiner rejected claims 6, 8, 11, 13, 19, 21, 24, 26, 32, 34, 37 and 39 under 35 U.S.C. § 103(a) as being unpatentable over Lee in view of Calusinki (U.S. Publication 2005/0071342). Applicant traverses the rejection for at least the following reasons.

With respect to the § 102 and § 103 rejections, Applicant also asserts that numerous ones of the dependent claims recite further distinctions over the cited art. However, since the rejection has been shown to be unsupported for the independent claims, a further discussion of the dependent claims is not necessary at this time. Applicants reserve the right to present additional arguments.

CONCLUSION

Applicants submit the application is in condition for allowance, and an early notice to that effect is requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5681-72300/RCK.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Applicant(s)

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: May 30, 2008